Version 1.0
February 17, 1998

STAR Note 325
STAR Table Structures for TPC Data Distribution

R. Bossingham
Nuclear Science Division
Lawrence Berkeley National Laboratory

Two formats, appropriate for STAR TPC data distribution by an event server, are defined. The formats consist of STAF tables within a directory structure: one relates to the TPC pad planes directly; the other relates to their instrumentation. Data subsets are easily extracted for distribution at the level of a half pad row or FEE card, respectively.

The relationships of the data structures to the on-line event server, as well as off-line cluster/hit-finding, are explored. The design has evolved to the point that both should be able to use the structures efficiently, but event server requirements must be specified before the design can be finalized. The structures will be more efficient than the existing ones with the **tcl** cluster/hit finder, but a **tcl** rewrite will be needed to exploit them fully.

The formats evolved from those used for TPC cosmic-ray testing,[1] but differ enough as to be incompatible. The new formats better support the 24 TPC sectors, optionally index the data at finer (pad-level) granularity, are more appropriate for an event server, and eliminate compromises made due to MiniDAQ constraints. They are also more weighted toward efficient representation of central Au-Au events, and their storage overhead for such events is a bit less, when indexed at the same level. Recent **Venus+tss** simulation results[2] predict an overhead of ~21% for such events. The ~61 KB/event overhead for sparse events (e.g., peripheral collisions) is acceptable in absolute terms, but represents a much higher percentage (~285%).

Introduction

STAR data will be transmitted and recorded by DAQ in a compressed format that is opaque until translated by DAQ-supplied library routines. The translation will apparently occur both online, in a system connected to the event server, and offline, after events are read from tape. If the same TPC software is to be used online and offline, the translated TPC format must be able to

[1] R. Bossingham, STAR Note SN0282 (1997).

[2] Iwona Sakrejda, http://www.rhic.bnl.gov/star/starlib/doc/www/html/tpc_l/tpc.html .

accommodate the needs of the on-line event server and diagnostics, as well as the off-line analysis. It must also support the full range of TPC data densities, from single cosmic rays to central Au-Au collisions to pedestal events.

An appropriate format for the translated TPC data has not yet been defined. A candidate, derived from the TPC cosmic-ray testing format (referred to here as the *SN0282 format*), is proposed in this note. The SN0282 format, itself, is not ideal for the representation, distribution and analysis of data from the full TPC; it was primarily designed for efficient data storage, with compromises made to accommodate MiniDAQ limitations. Some shortcomings of this older format will be mentioned briefly in the next section.

Other STAR collaborators are responsible for some aspects of the SN0282 and currently proposed formats. Iwona Sakrejda suggested a number of specific improvements to the format and this note. Nathan Stone and Mark Gilkes wrote translation codes that revealed some awkward features inherent in the SN0282 format. Susanna Jacobson discussed and tested several aspects of the SN0282 format, and, in particular, determined requirements related to word-boundary alignment. Craig Tull showed some of the potential of Staf with an earlier, informal data-format proposal. More generally, TPC data formats have been discussed at length by Mike Levine, Doug Olson, Lanny Ray, Jo Schambach, Craig Tull, Bill Greiman and others.

TPC Data Format Considerations

Data from the STAR TPC can be structured in two natural ways:

- Physical subdivisions: sector, pad row, pad and time bucket. To reduce granularity, variable lengths and the data table sizes, the proposed format also separates inner and outer subsectors; first and second halves of pad rows; early and late parts of the time-bucket range. Calibration and analysis values apply mostly at the subsector, pad and pixel levels.

- Electronics subdivisions: sector, readout board, Front-End Electronics (FEE) card, FEE channel and time bucket. The proposed format divides the data by readout board (1–6); and, again, by early and late parts of the time-bucket range. Calibration and analysis values apply at the sector, readout board, FEE card, FEE channel, and pixel levels, with occasional references to FEE chips and the 16-channel segments grouped by the readout boards.

Information associated with these subdivisions includes:

Sector, subsector: ID; geometrical alignment; availability of data (DAQ will support sector-level granularity); lists of pad rows with data entries.

Pad row: Cluster and hit finding in **tcl** is by pad row, and some calibration effects could appear at this level (ground, anode and field wires span a pad row). Pad-row values include row ID, lists of pads with data entries, and row geometry (number of pads and location).

Readout board (RDO): RDO ID; status read by from Slow Controls; FEE location list; FEE cards with data entries; trigger-time offset; RDO slow-controls values.

FEE card: FEE card ID (both RDO and sector locations); channels with data entries.

Pad/channel: ID (location within a row); bad channels; t_0 calibration; time buckets with data entries; FEE pulser status; and fixed-length lists of ADC, pedestal, or noise data, if a standard time-bucket range was defined.

Time bucket: ID; ADC value (with/without pedestal subtraction and gain correction; linear, non-linear or truncated; 8-bit, 10-bit or floating); mean pedestal; rms; drift of pedestal mean.

Beyond the need to parallel the TPC structure, the format is shaped by practical considerations. Many of these differ from those relevant to MiniDAQ that shaped the SN0282 format. Current design considerations include these:

- Structures must be defined to support up to 24 sectors, 2 subsectors per sector, 45 pad rows per sector, 182 pads per pad row and 512 time buckets per pad.
- Structures must be defined to support up to 24 sectors, 6 readout boards per sector, 36 FEE cards per readout board, 32 channels per FEE card and 512 time buckets per channel.
- A fixed number of time buckets per channel (i.e., 512) *cannot* be assumed; the ADC sampling frequency may be reduced, subdividing the TPC drift volume into fewer time buckets ($\sim$300).
- The format need not support modified FEE's for anode-wire readout (supported by MiniDAQ—but not actually used). Anode-wire hits will now be recorded through trigger.
- The format should emphasize convenience and speed of access, even at the expense of compactness, since it is not primarily intended for bulk data storage.
- Generalized structures are needed, allowing an event server to provide subsets of the data, and/or data types, in similar form to all data consumers.
- The format must allow efficient data-subset extraction at the granularity to be supported by the event server (ideally, pad row, FEE card or less) with only minimal decoding.
- Data should be efficiently accessible for cluster and hit-finding at the granularities that these codes need: subsector, pad row and pad.
- Structures should be compatible with those associated with cluster and hit finding.
- The somewhat random relation of FEE channels to TPC pads excludes an efficient structure that directly references single data tables by both pads and FEE channels. However, indirect referencing can, and must, be supported, using information from the data stream.

- The format must support the full range of STAR data densities. This implies flexibility, both event by event (runs will mix triggers types) and within an event. Format decisions must be based on simple criteria (e.g., data density.
- The Staf-level directory scheme should segregate tables by sector, allowing parallel structures for each. Processing is at, or below, the sector level until track segments are linked across sectors. (MiniDAQ's single-level sub-directories preclude much sophistication.)
- Data for cluster finding should not split a pad row between tables. (MiniDAQ's i960's cannot access data from more than two readout boards, so data are stored in six distinct table sets, with pad rows sometimes split between sets).
- The data should be ordered and grouped so that memory accesses tend to be sequential.
- Table sizes should be consistent with efficient processing; small tables have too much overhead, and large tables can exhaust computer memory. Tables should hold the amount of data handled in one stage—usually, that from one subsector or one readout board.
- Table columns should hold a single kind of information, and rows should map to a single entity (e.g., a pad row, pad or pixel). Adding tables should not exact a large performance penalty (unlike MiniDAQ), so kluges to squeeze data into unnatural places can be avoided.
- The format should provide explicit flags and values for decoding. (SN0282 defined 22 formats, each implying a set of decoding decisions; this complicated decoding.)
- Decoding flags should be placed in the table hierarchy above the level where they are applied, but not so high as to unnecessarily compromise generality.
- The naming scheme for directories and tables should be mneumonic, consistent and extensible.
- Table definitions must avoid memory mis-alignment: the row length must be a multiple of the longest data type's length, and the length of data types cannot increase across a row, and table lengths should be a multiple of the word length.

Nomenclature

Glossary

Black data: Data that includes every pixel within the allowed range.

DAQ: Data-AcQuisition (the system, or the group responsible for the system).

FEE: Front-End Electronics; each card has 32 channels shared by two SAS/SCA chip pairs.

Grey data: Data that includes at least one pixel for every pad within the allowed range.

Mapped: Mapped data structures reflect the physical TPC padplanes; generally intended for cluster- and hit-finding, and, possibly, tracking.

Native (or unmapped): Native data structures reflect the TPC instrumentation and are intended for studying and debugging the electronics.

RDO: Readout board. Services up to 36 FEE cards; reads digitized data and sends it to DAQ.

SAS: Amplifier/Shaper. Two SAS chips form each FEE card's front end.

SCA: Switched Capacitor Array: Two SCA chips digitize data on each FEE card.

sector_ : One of the 24, 30-degree TPC **sector**s on one side of the central membrane. Each consists of two subsectors, and is instrumented by six readout boards.

Sparse data: Data lacking entries for enough pads (more than ~30%) that it is more efficient to mark pads with, than without, data.

Directory Naming

This note involves only the TPC, but Staf directory names must anticipate directories for other detectors; simplistic directory names (e.g., "DATA") must be avoided. An obvious solution is to prepend "TPC_" to all TPC base-level directories. Further, since TPC directories will exist for data, calibrations, global tracking and so forth, names should not be too narrow. For example, the directories for raw data may also store cluster and hit tables, so "TPC_DATA" is preferable to "TPC_RAW_8_BIT_ADC_DATA."

The directory names used in the figures for this note are "TPC_DATA," for raw and processed data, and "TPC_CAL," for some types of calibration data. However, such naming conventions are completely divorced from the substance of the proposed format.

Table Naming

A table-naming convention provides fairly short, descriptive names for our tables, while allowing the addition of more tables with names of the same form later (for cluster and hit-finding, perhaps). However, many tables contain several types of qualitatively different information, making some names neccessarily vague or incomplete. (Segregating every different type of information into a different table would cause the number of tables to proliferate inconveniently.) Table names contain up to four parts: *StructureType_RowEntity_TableContentType_SectorSubdivision*.

Possibilities for *StructureType* include:

raw_ : Data *structured* like raw ADC data (e.g., pedestal means and gains).
cls_ : Clusters of pixels and related information.
rcl_ : Cluster information relating to raw data only.
hit_ : Hits and related information.

RowEntity means the thing described by one row in the table:

sector_ : One full **sector** within the TPC.
row_ : A pad **row** within a sector.
pad_ : A **pad** within a pad row.
seq_ : A **seq**uence of consecutive time buckets from a pad (or FEE channel).
fee_ : A Front-End Electronics (**FEE**) card; up to 36 plug into a readout board.
chn_ : A **ch**a**n**nel on a FEE card.
grd_ : A grid element with Cartesian axes defined by one pad row and z.

TableContentType partially defines the information in the table for each *RowEntity*:

des_ : A **des**cription of lower-level tables, if no more specific type applies.
idx_ : One **i**n**d**e**x** (or more) points to lower-level table rows; additional information may appear.
id_ : An **ID** for this entity (e.g., a pad or FEE-channel number).
nin_ : The **n**umber of entries **in** a lower-level table for this entity.
off_ : One, or more, index **off**set, pointing into a lower-level table.

The *SectorSubdivision* is the part of a sector to which a table refers.

in : Inner subsector.
out : Outer subsector.
ri : Readout board i; $i \in [1, 6]$.

Conventions for Table Entries

Referencing If some entity might be associated with a lower-level table row, three referencing options are often available:

N : None: The lower-level table does not exist.
R : Range: ID's from an entire, defined range are associated in order.
L : ID's are explicitly listed in a table.

Format Two basic formats are supported:

U : Unmapped, or native. Data are referenced to TPC readout boards and FEE cards.
M : Mapped. Data are referenced to the TPC padplane and its components.

Variable types Table entries are defined according to a semi-standard convention:

char : Character or signed octet.
u_char : Unsigned octet.
short : Signed, 2-byte integer.
u_short : Unsigned, 2-byte integer.
long : Signed, 4-byte integer.
u_long : Unsigned, 4-byte integer.

Data Format Definition

Format Overview

The data structures are defined with access, extraction and storage efficiency in mind. They are intended to be "natural," exploiting STAF's capabilities—not just bags o'bytes. Two distinct sets of structures are defined—*mapped* and *native*—along with flags to define, in part, their interpretation. Pure (often pad- or pixel-level)data are segregated into independent tables.

- Mapped data structures reflect the physical TPC padplanes and are intended for hit-finding and tracking. Figure 1 shows the basic scheme. The DAQ data stream is to contain the information needed to map data onto pads, without reference to external maps; this should not change during a run. Note that data from "blind" FEE channels (those unassociated with a pad) cannot be represented.

- Native-data structures reflect the TPC instrumentation and are intended for electronics studies and debugging. Figure 2 shows the basic scheme. FEE-channel data trivially map to IC chips or 16-channel groups, as needed. FEE channel maps *can* relate native-format data to the physical TPC, but this is not efficient, unless forced by simultaneous electronics and tracking studies. (A graphics display to allow direct viewing of native data would be very valuable—a readout-board analogue to the Pad Monitor that displays pad-plane data.)

The tables used to represent the data in mapped and native formats are listed in Table 1. They will be defined and discussed below. Within this framework, adding more parallel data tables does not increase the overhead; and data table deletion requires no changes to other tables. However, adding or deleting items *within* a data table can force significant changes. In particular, it is expensive to interleave sets of tables, as would be neccessary if the data from each readout board, or each i960, were translated as an independent table. Instead, translation and merging should happen in one step as the derived tables are first filled.

Sector Description

Different TPC sub-directory branches can be created to hold different classes of information; obvious ones include calibration data and event data. A branch is described, to a large extent, by its

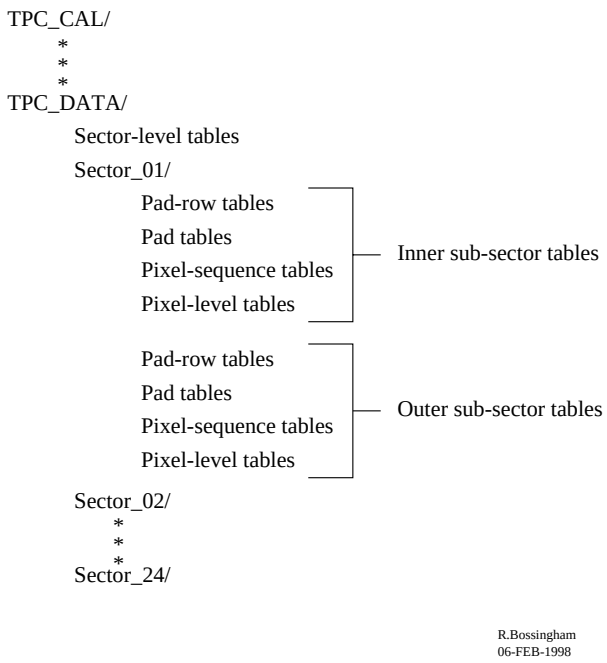

Figure 1: Overall structure to store mapped data, showing TPC data and calibration branches. TPC_CAL and TPC_DATA have the same substructure.

Hierarchy Level	Mapped Data Tables (*=**in**, **out**)	Native Data Tables (*=**1**,**2**,**3**,4,**5**,**6**)
Full TPC	**raw_sector_des**	**raw_sector_des**
Sector_N/ Level 1	**raw_row_idx_***	**raw_fee_idx_r***
Sector_N/ Level 2	**raw_pad_nin_***	**raw_chn_nin_r***
”	**raw_pad_off_***	**raw_chn_off_r***
”	**raw_pad_id_***	**raw_chn_id_r***
Sector_N/ Level 3	**raw_seq_des_***	**raw_seq_des_r***

Table 1: List of table names defined in this note. The tables, listed side by side for the mapped and native formats, are similar and serve the same purposes.

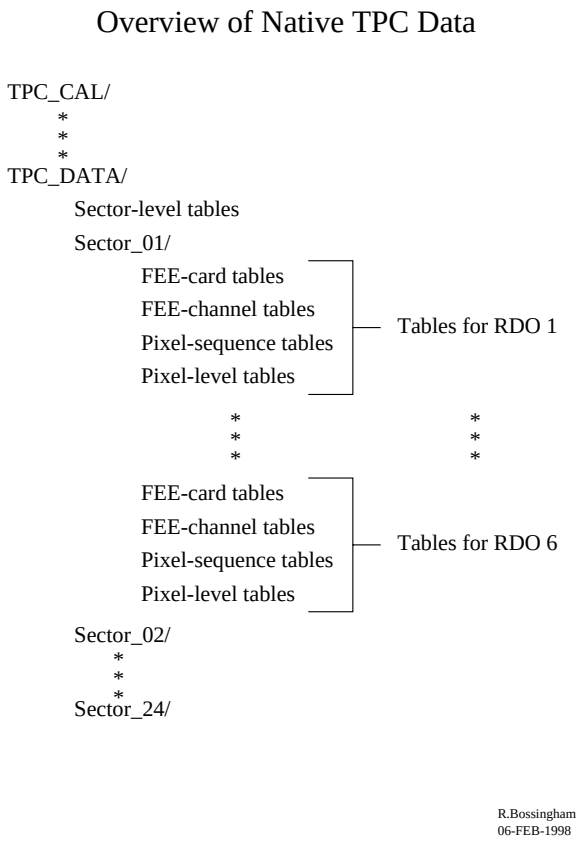

Figure 2: Overall structure to store native data, showing TPC data and calibration branches. TPC_CAL and TPC_DATA have the same substructure.

raw_sector_des table, having exactly 24 rows and ordered by sector ID (1–24).

Each row declares the data for that sector as mapped or native, and gives time-bucket information. For mapped TPC data, it describes pad-row tables by subsector; for native data, it describes the FEE-card tables by RDO. The contents are defined in Table 2. Its 864-byte length is usually negligible, compared to the event size; a more compact form would save little space.

Two implicit assumptions influence this format definition:

1. Time-bucket ranges and zero suppression are set at the sector level or above (i.e., globally).

2. The method of associating pad ID's with data *can* change below the sector level. (Row-by-row determination may improve storage efficiency.)

Tables parallel to **raw_sector_des** could be useful to the event server, marking, perhaps, which data tables exist. Such a table should not be used by the analysis chain, since it defines a fixed list of possible tables, and new, parallel tables will be needed as the software develops.

At the same sub-directory level as the **raw_sector_des** table are the 24 (possibly empty) sub-directories **Sector_N/** for the sectors; **raw_sector_des** describes their highest level tables. All of the tables are discussed and defined below. The semi-hierarchical table relationships for mapped and native data are shown schematically in Figs. 3 and 4, respectively, and these figures may be helpful in understanding the discussions of the tables.

Pad-Row or FEE Description

For mapped data, pad-row descriptions are given by the **raw_row_idx_*** tables, defined in Table 3. They allow one to jump into the pad, sequence and pixel tables with half-pad-row granularity, The entries that give the number of rows in the sequence- and pixel-level tables for both halves of the pad row are, strictly speaking, redundant, but ease extraction of half-pad-row data subsets.

Half-pad-row granularity was chosen to improve random access within pad-level tables, allow extraction of small data subsets by the event server, and allow pixel-data offsets within that half pad row to be held in a short int variable. (Optional **raw_pad_off_*** tables hold such offsets, as discussed below.)

For native data, the **raw_fee_idx_*** tables, defined in Table 4, describe FEE-card-level tables at FEE-card granularity and provide row indices into the FEE channel, pixel-sequence and pixel tables. The table definition resembles that for mapped data. The number of rows in the lower-level tables for each FEE card is given to allow one to extract the lower-level tables more easily.

Declaring Pad or Channel ID's There are three options for declaring the pads or channels represented in subsequent data tables: 'N' for no entries at this level, 'R' for a single, unbroken range, and 'L' for an explicit, ordered list. When data are sparse, this last option becomes efficient—both in terms of access and storage. The **raw_pad_id_*** tables list and order pads by number within a pad row; the **raw_chn_id_*** tables list and order channels by number within a FEE card. Tables 5 and 6 give descriptions.

Entries within Row	Values	Type	Granularity	Bytes
First time bucket	1 – 512	u_short	Sector	2
Last time bucket	1 – 512	u_short	”	2
“Middle” time bucket	1 – 512	u_short	”	2
Time-bucket referencing	N, R, L	char	”	1
Format: unmapped, mapped	U, M	char	Sector	1
Pad-row entries	0 – 13	u_char	Inner subsector	1
First pad row	13 – 1	u_char	”	1
Pad-row referencing	N, R, L	char	”	1
Pad-row entries	0 – 32	u_char	Outer subsector	1
First pad row	45 – 14	u_char	”	1
Pad-row referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 1	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 2	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 3	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 4	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 5	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
FEE-card entries	0 – 36	u_char	RDO 6	1
First FEE card	1 – 36	u_char	”	1
FEE-card referencing	N, R, L	char	”	1
Unused		3 × u_char		3
Sector ID	1 – 24	u_char	Sector	1
Total bytes per row				36

Table 2: Row description for the 864-byte, fixed-length **raw_sector_des** table. One 36-byte row per sector, ordered by sector ID from 1–24, defines the interpretation of subsequent tables.

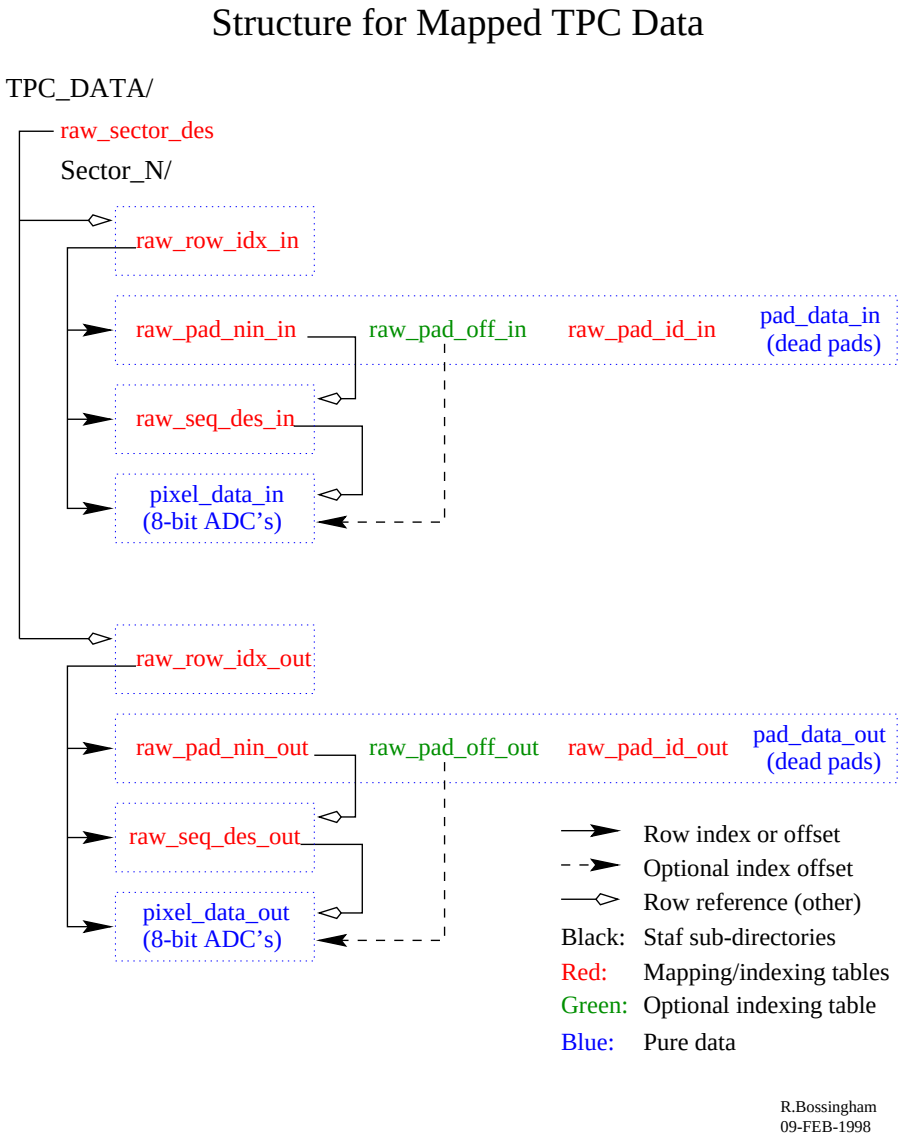

Figure 3: Structure for mapped TPC data. Arbitrarily chosen data tables are shown, but the structure applies as well to calibrations. The tables are discussed in the text.

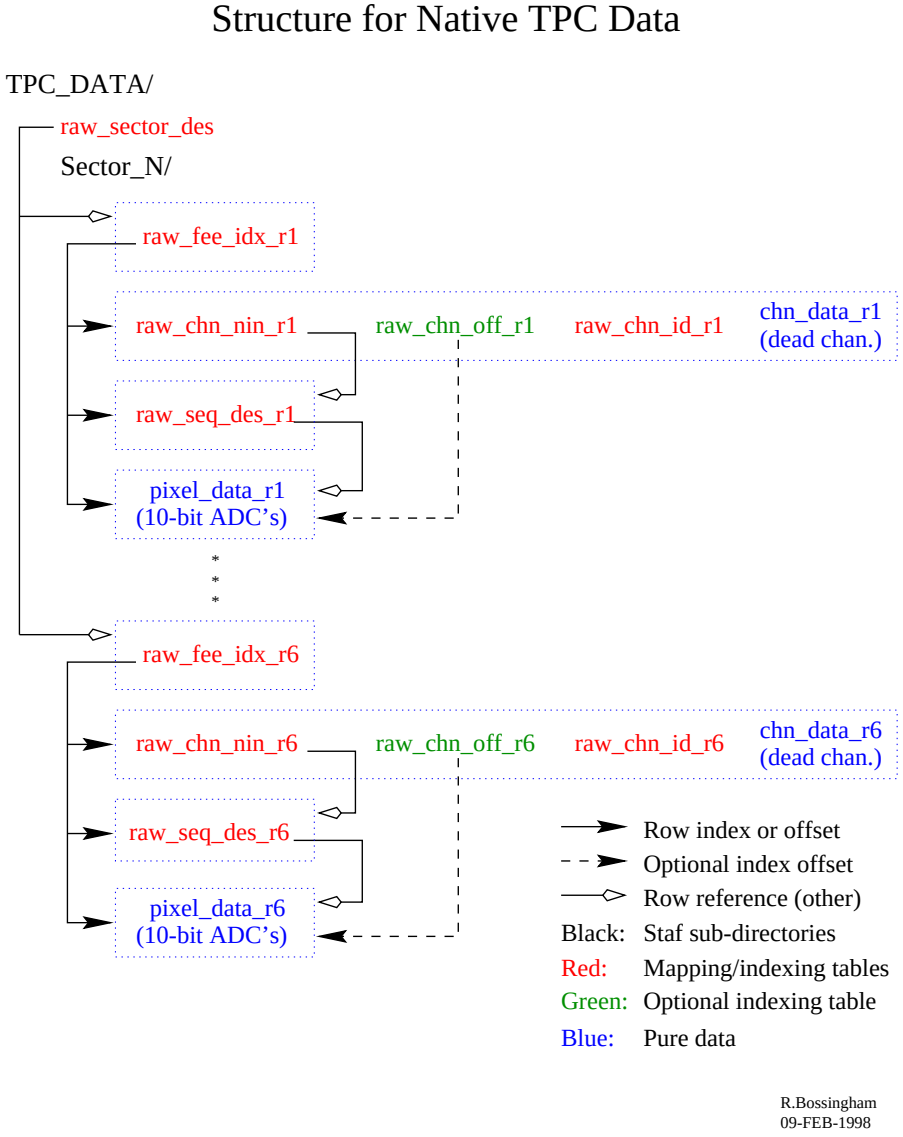

Figure 4: Structure for native TPC data. Arbitrarily chosen data tables are shown, but the structure applies as well to calibrations. The tables are discussed in the text.

Entries within Row	Values	Type	Granularity	Bytes
Index, pixel-data tables	0 – ∼2,018,000	u_long	Pad row	4
Index, **raw_seq_des_*** table	0 – ∼2,010,000	u_long	”	4
Index, pad-data tables	0 – ∼3942	u_short	”	2
Pixel-table entries	0 – 46,410	u_short	Half pad row 1	2
Sequence table entries	0 – 46,592	u_short	”	2
Pixel-table entries	0 – 46,410	u_short	Half pad row 2	2
Sequence table entries	0 – 46,592	u_short	”	2
Pad table entries	1 – 91	u_char	Half pad row 1	1
First pad in range	1 – 91	u_char	”	1
Pad table entries	1 – 91	u_char	Half pad row 2	1
First pad in range	45 – 182	u_char	”	1
Pad referencing	N, R, L	char	Pad row	1
Pad row ID	45 – 1	u_char	”	1
Total bytes per row				24

Table 3: Row definition for the **raw_row_idx_in**, **_out** tables, representing mapped data from one pad row in a subsector. Rows are inverse ordered by pad-row ID (13–1 and 45–32). Giving the number of table entries for the first and second equal halves of a pad row separately allows one to extract, or jump into, the lower tables at half-pad-row intervals more easily. Pad ID's are numbered from the start of the full row, as usual.

Entries within Row	Values	Type	Granularity	Bytes
Index, pixel-data tables	0 – ∼2,358,800	u_long	FEE card	4
Index, **raw_seq_des_*** table	0 – ∼2,349,600	u_long	”	4
Index, channel-data tables	0 – ∼4607	u_short	”	2
Pixel table entries	0 – 16,320	u_short	FEE card	2
Sequence table entries	0 – 16,384	u_short	”	2
Channel table entries	0 – 32	u_char	FEE card	1
First channel	1 – 32	u_char	”	1
Channel referencing	N, R, L	char	”	1
FEE card's location on sector	0 – 181	u_char	FEE card	1
FEE card's location on RDO	1 – 36	u_char	”	1
Unused		u_char		1
Total bytes per row				20

Table 4: Row description for the **raw_fee_idx_r1–_r6** tables, representing the native data from one FEE card. Rows are ordered by increasing FEE card location number (1–36) on a readout board. They resemble the **raw_row_idx_*** tables.

Row Entry	Values	Type	Granularity	Bytes
Pad within pad row	1 – 182	u_char	Pad	1

Table 5: Row description of the **raw_pad_id_in**, **_out** tables for mapped data; one pad per row. These are required if pads within a row are declared by **raw_row_idx_in** to be listed explicitly. They are efficient for sparse data.

Row Entry	Values	Type	Granularity	Bytes
Channel within FEE	1 – 32	u_char	Channel	1

Table 6: Row description of the **raw_chn_id_r1–_r6** tables for native data, one channel per row. These are required if FEE-card channels are declared to be listed explicitly in the corresponding **raw_fee_idx_*** . They are efficient for sparse data.

If a large fraction of the pad or FEE channels have entries, it is more efficient to note the ones without data, than those with it. Two considerations in optimizing are:

1. Savings from efficient random access to pad or channel data when the offset within a table is known, versus the cost of accessing empty entries. For mapped data, indexing is at half-pad-row intervals—63 pads, on average. If the optional, pad-level indexing for the pixel data is provided, a few look-ups can locate a pad and its pixel data by successive interpolation. However, if zero-suppressed pixel data are indexed *only* by half pad row, all sequence lengths preceding the requested one in the half pad row must be systematically summed, and pixel-data access must be done sequentially.

2. Savings from not listing pad or channel ID's, versus the cost of storing empty entries. For mapped pixel data, a pad-list entry is one byte, and zero-length sequence entries are two bytes. If no other pad-level tables (e.g., pad-level indexing) are used, break-even occurs when 2/3's of the pads have data.

Therefore, subsectors or RDO's with at least ∼70–90% pad occupancy can be treated as "grey" (i.e., having data on all pads, but not all pixels). Unless the data format is used for bulk data storage, for which percent of overhead costs serious money, more precision is unnecessary.

Pixel Data Description

If pixel entries fill a defined range, as for black data, the above tables provide all information needed for random access to any pixel. In the simplest case—that STAR defines a standard time-bucket range—all pixel data for a pad or channel can go into one row of a table that parallels other pad or FEE-channel data. However, the time-bucket description becomes non-trivial for zero-suppressed data, and this will be the usual case for STAR data-taking.

Representation of Arbitrary Time-Bucket Sequences

Time-bucket sequences on a pad can be represented by giving the number of sequences, along with the length and first time bucket of each one. Support for more than 256 time buckets complicates access and storage, since a trivial representation with octets is excluded. The scheme here (generalized from the SN0282 format), divides the time-bucket range into two parts, of ≤ 256 time buckets each; their lengths need not be equal (but probably would be). The first, last and "middle" time buckets are declared in **raw_sector_des**. The following discussion refers only to pads, but applies equally to FEE channels. These variables are used:

M : Number of sequences starting in part one of the pad's time-bucket range; 0–255.
i_k : For a part-one sequence, the number of time buckets *after* the first; 0–255.
m_k : For a part-one sequence, the first time bucket's offset from the start of part one; 0–255.
N : Number of sequences starting in part two of the pad's time bucket range; 0–255.
j_k : For a part-two sequence, the number of time buckets *after* the first; 0–255.
n_k : For a part-two sequence, the first time bucket's offset from the start of part two; 0–255.

M and N, the number of sequences on a pad, are stored as pad-like data in the **raw_pad_nin_*** tables described in Table . These tables are required if **raw_sector_des** declares time-bucket sequences for a subsector to be listed explicitly. $M = N = 0$ is legal and used to mark pads without pixel data. The $M, N \leq 255$ limits exclude only the extreme case of 256 consecutive, one-bucket sequences.

Column 1	Column 2	Type	Bytes
M	N	$2 \times$ u_char	2

Table 7: Row description for the **raw_pad_nin_in** and **_out** tables for mapped data, and the **raw_chn_nin_r1**–**_r6** tables for native data Each row gives M and N for one pad or FEE channel.

M and N will vary widely, so the remainder of the sequence descriptions are given as an ordered list in the **raw_seq_des_*** tables described in Table . Each row gives the length and first time bucket of one sequence. These tables are required if **raw_sector_des** declares sequences to be listed explicitly. For zero-suppressed data, sequences are usually short and disjoint, but do not *have* to be; in principle, 512 consecutive time buckets can be represented as two, 256-bucket sequences.

Redundant Indexing Tables

The pixel data are completely defined at this point, but random access to a pad's pixel data is slowed by the fact that it is only indexed at half-pad-row intervals. Therefore, the format allows redundant tables that give index offsets (or, in principle, complete indices) at finer granularity. Other tables are unaffected. Two pairs of tables should be of value:

- For mapped data, the **raw_pad_off_*** tables give offsets, at the pad level, to the sequence and pixel indices in **raw_row_idx_***.

Column 1	Column 2	Type	Bytes
m_1	i_1	2 × u_char	2
m_2	i_2	2 × u_char	2
.	.	.	.
.	.	.	.
.	.	.	.
m_M	i_M	2 × u_char	2
n_1	j_1	2 × u_char	2
n_2	j_2	2 × u_char	2
.	.	.	.
.	.	.	.
.	.	.	.
n_N	j_N	2 × u_char	2
Total bytes for pad			$2 \cdot (M + N)$

Table 8: Section of a **raw_seq_des_in**, **_out** table, listing sequences on one pad; or of a **raw_seq_des_r1–_r6** table, listing sequences on one FEE channel. Each row gives the length (minus one) and starting position of one sequence.

- For native data, the **raw_chn_off_*** tables give offsets, at the channel level, to the sequence and pixel indices in **raw_fee_idx_***.

They are defined in Tables 9, 10 and allow random access to the pixel data for any given pad or FEE channel. Their use is not specified or implied by any other table. Still, if they exist with non-zero row count, all random accesses to zero-suppressed pixel data would normally use them.

Column 1	Values	Column 2	Values	Type	Gran.	Bytes
Index offset: pixel data	0 – ∼46,080	Index offset: **raw_seq_des_***	0 – ∼45,090	2 × u_short	Pad	4

Table 9: Row description for the **raw_pad_off_in**, **_out** tables, with one row per indexed pad. These tables are optional and redundant.

Column 1	Values	Column 2	Values	Type	Gran.	Bytes
Index offset: pixel data	0 – ∼46,080	Index offset: **raw_seq_des_***	0 – ∼45,090	2 × u_short	Chan.	4

Table 10: Row description for the **raw_chn_off_1–_r6** tables, with one row per indexed channel. These tables are optional and redundant.

The tradeoff in CPU time is between that spent creating extra tables, versus that saved during data accesses. On average, random access must go through half of the sequence descriptions

between two pointers, so two random accesses to dense data cost nearly the same as calculating index entries for all pads within this range. Half pad rows average 63 pads, so creating the extra table is economical if more than about 2/63=3% of the pads will be randomly accessed.

Data Discussion

Pad and Channel Data

Excepting pedestals, most characteristics of the TPC electronics (gain, t_0 offset, time constant, bad or saturated channels, FEE channel mapping onto the pad plane, etc.) can be described by one or more values that apply to all time buckets for one pad (or FEE channel). Multiple entries in parallel and/or multi-column tables can apply to each pad or channel.

A special type of pad/channel data is the 10-to-8-bit lookup table for the DAQ ASIC's; this would usually be calculated from scalar information (e.g., gain), and one would neither want, nor need, to export it. However, the capability might be necessary for debugging. Collectively, these 1024-entry tables are quite large, but relatively squishy (a non-lossy algorithm achieving 19:1 compression was tested).

Pixel Data

Pixel-data tables, described by sequence lists, must allocate one row per time bucket in a sequence, but their form is otherwise unconstrained. The tables are interpreted by starting from an indexed row and calculating the total offset to some pixel using the M, m_k, N and N_k entries. This is expensive if sequences are randomly accessed and indexed rows are widely spaced, which motivates support for redundant tables with pad-level indexing.

The primary pixel data are ADC values or their processed derivatives, but any type, or mixture types, supported by Staf is allowed in a row: signed or unsigned; byte, integer or floating. Row-length and data-type ordering is constrainted somewhat by requirements on memory alignment. Structures created by well-defined analysis tasks, where flexibility is not an issue, might create multi-column, mixed-type tables, but raw ADC's and other pixel data from DAQ should be placed into one-column tables to allow selectivity in passing and retaining data.

Pedestal means, etc. calculated on the DAQ i960's will presumably be provided as integer values, possibly scaled to give sub-integer precision. (MiniDAQ scales pedestal means, shifts, and rms's by 8 and rounds to the nearest integer, limiting truncation errors to $\pm 1/16$th count.) On the other hand, pixel data calculated offline may be floating point.

In MiniDAQ, raw, ten-bit ADC values are stored as u_short's, and, if pedestal subtracted, are constrained to be non-negative. This is usually desirable, but signed values, after pedestal subtraction, would sometimes be extremely useful. Pedestal shifts are passed as short int's (*signed*, of course).

TPC ADC's are 10-bit, with mean pedestals of 50–255 (typically, 200); pedestals occasionally fall outside the allowed range due FEE defects, shorted pads, induced signals, etc. This data

may be suppressed or ignored, but must be handled gracefully. Without pedestal subtraction, the maximum ADC value is determined by clock counts within the SCA digitization window: $\sim$1015 (*not* 1023), but not fixed precisely. If gain corrected, this limit does not apply.

MiniDAQ supports two distinct types of u_char ADC values: one translates 10-bit ADC's with a non-linear lookup table; the other constrains ADC's to ≤ 255, and truncates them to 8 bits (for pedestal values, intended to be ≤ 255).

Interpreting pixel data requires additional information concerning pedestal subtraction; gain correction; threshold and other parameters, if zero suppressed; bit patterns designating overflows; etc. The form of this information is not defined here. Table-naming conventions, parameter tables, external databases or something else altogether might be used.

Readout Board Data

Tables from specific readout boards are more directly connected to native-format data, but might, sometimes, be appropriately stored with mapped data structures. The format definition does not require *all*, data tables in a sub-directory to have parallel structure. However, for sanity's sake, the sector sub-directories should parallel one another, and their tables should refer to that sector; other tables belong elsewhere.

The sub-directory for a sector might, for example, hold the six header tables from a sector's six readout boards for an event. These allow certain checks; a less-than-final form of the header is shown in Table 11.[3] In contrast, putting 144 tables from all readout boards on the TPC into, say, the sector 1 sub-directory would clearly violate the philosophy of the structure.

Non-local Sector Information

There is no strict requirement that tables within the sub-directory for a sector contain only information that is local to that sector, and such a requirement is probably not practical. For example, after global TPC tracking, track segments may connect across sector boundaries, so tracks are no longer local to a sector. Nonetheless, if a table associating pixels with tracks were created, it should be placed in the sector sub-directory, though the table of global tracks itself should be elsewhere.

Storage Overhead

Central Au-Au Collision Overhead ("Standard" Assumptions)

As said before, the proposed format emphasizes ease of use and access efficiency over storage efficiency. However, **tss** simulator output might be stored in this format, and it also has implications for computer memory requirements, so an estimate of storage overhead is appropriate. We take assumptions from previous data-format discussions, adding others as needed. For central Au-Au events we assume:

[3] *Front-End Electronics to DAQ Receiver Board Fiber Optics Interface Specification Version 1.0a*, V. Lindenstruth et al., and from discussions with Fred Bieser.

Entry	Bits	Definition
0	4	trigger CMD
1	4	DAQ CMD
2	4	trigger token
3	8	trigger token LSW
4	8	bunch crossing counter
5	4	detector ID
6	8	readout unit ID
7	8	spare
8	8	spare
9	8	spare
10	8	spare
11	8	spare
12	8	tagword ($de)
13	8	tagword ($ad)
14	8	tagword ($fa)
15	8	tagword ($ce)
16	8	FEE 1 location ID (maybe)
.	.	.
.	.	.
.	.	.
51	8	FEE 36 location ID (maybe)
52	-	detector specific (undefined)
	.	.
	.	.
	.	.
63	-	detector specific (undefined)

Table 11: Possible event header from a readout board; definition not yet finalized.

- $N_{\rm seq} = 4$ sequences per pad.
- $L_{\rm seq} = 8$ time buckets per sequence.
- No pad-level indexing tables.
- No **raw_pad_id_*** tables (since data are "grey").
- Negligible Staf-level table overhead.
- 8-bit ADC data.

These assumptions are chosen mostly for consistency; others, probably more realistic, will be tried below.) With 1080 pad rows and 136,560 pads, 8-bit TPC ADC data consumes

$$L_{\rm seq} N_{\rm seq} N_{\rm pads} = 8 \cdot 4 \cdot 136,560 = 4,369,920 \text{ bytes} ;$$

and the 32% storage overhead consists of:

1. The fixed-length **raw_sector_des** table: 864 bytes.
2. 24 **raw_row_idx_*** table pairs: $24 \times (13 + 32) \times 24$ bytes $= 25,920$ bytes.
3. 24 **raw_pad_nin_*** table pairs for 5690 pads: $24 \times 5690 \times 2$ bytes $= 273,120$ bytes.
4. 24 **raw_seq_des_*** table pairs for $N_{\rm seq} \times N_{\rm pads} = 4 \times 5690 = 22,760$ sequences: $24 \times 22,760 \times 2$ bytes $= 1,092,480$ bytes.

Central Au-Au Collision Overhead (Venus + tss Simulation Assumptions)

Recent simulations of central Au-Au events[4] using **Venus** and the current (early 1998) **tss** version obtained a much greater hit density than assumed above:

- $N_{\rm seq} = 12$ sequences per pad (average).
- $L_{\rm seq} = 10.5$ pixels per sequence (average).

Retaining the other assumptions, the 8-bit ADC data now consumes

$$L_{\rm seq} N_{\rm seq} N_{\rm pads} = 10.5 \cdot 12 \cdot 136,560 = 17,206,560 \text{ bytes} ;$$

and the 21% storage overhead consists of:

1. The fixed-length **raw_sector_des** table: 864 bytes (always).
2. 24 **raw_row_idx_*** table pairs: $24 \times (13 + 32) \times 36$ bytes $= 25,920$ bytes (as before).

[4] Iwona Sakrejda, http://www.rhic.bnl.gov/star/starlib/doc/www/html/tpc_l/tpc.html .

3. 24 **raw_pad_nin_*** table pairs for 5690 pads:
   $24 \times 5690 \times 2$ bytes $= 273,120$ bytes (as before).

4. 24 **raw_seq_des_*** table pairs for $N_{\mathrm{seq}} \times N_{\mathrm{pads}} = 12 \times 5690 = 68,280$ sequences:
   $24 \times 68,280 \times 2$ bytes $= 3,277,440$ bytes.

Also, consider the cost of adding a redundant, pad-level indexing table; one four-byte row per pad adds $24 \times 5690 \times 4$ bytes $= 546,240$ bytes (3.2%) of storage overhead—a total of 24%.

Two-Track Event Overhead

Now consider a sparse event: two tracks with 90 minimum-ionizing hits, perhaps from a peripheral collision or cosmic ray. We assume:

- 45 real hits in each of two sectors.
- An additional 5% of the pads with 3-bucket noise sequences.
- Each real hit involves 3 pads ($= N_{\mathrm{wide}}$).
- $N_{\mathrm{seq}} = 1$ (one sequence per hit pad).
- $L_{\mathrm{seq}} = 5$ (five pixels per sequence for real hits).
- No pad-level indexing tables.
- **raw_pad_id_*** tables *are* used.
- Negligible Staf-level table overhead.
- 8-bit ADC data.

The 8-bit ADC data consumes

$$\sum_i L_{\mathrm{seq}_i} N_{\mathrm{pads}_i} = 5 \cdot (2 \cdot 45 \cdot 3) + 3 \cdot (0.05 \cdot 136,560) = 21,834 \text{ bytes} ,$$

and the 61-KB storage overhead consists of:

1. The fixed-length **raw_sector_des** table: 864 bytes (always).

2. 24 **raw_row_idx_*** table pairs:
   $24 \times (13 + 32) \times 36$ bytes $= 25,920$ bytes (as for Au-Au).

3. 7098 pad ID's in the **raw_pad_id_*** tables:
   7098×1 bytes $= 7098$ bytes.

4. 24 **raw_pad_nin_*** table pairs for a total of 7098 pads:
   7098×2 bytes $= 14,196$ bytes.

5. 24 **raw_seq_des_*** table pairs, representing a total of 7098 sequences:
   7098×2 bytes $= 14,196$ bytes.

Hierarchical structures are intrinsically inefficient for sparse events, and the situation is worse here because the granularity and pointer lengths were chosen to support dense data; a simple list describing all sequences in the TPC would have less overhead. The exact numbers are sensitive to threshold settings and other assumptions, but 61 KB (~285%) is a plausible guestimate for overhead.

Black Event Overhead

The overhead for black (e.g., pedestal) data is negligible ($< 0.1\%$), given the enormous data volume and its consistent structure. Any "reasonable" data format will do, and, in any case, relatively little non-zero-suppressed data should be stored in the future. So, while the proposed format has low overhead for black events, this is only a minor consideration.

How Squishy is the Proposed Format?

Overhead for central Au-Au events is of the most concern; the **raw_seq_des_*** tables completely dominate this. Several design choices allocate more bits to it than really necessary. If one restricts the format to 64, 16-bucket sequences per pad, and ignores word and byte boundaries, the **raw_seq_des_*** overhead shrinks from 21% to 16% (**Venus+tss** assumptions). Such savings are too small to justify the CPU cost and coding complications for an analysis format.

However, more general encoding/decoding schemes (e.g., Lempel-Ziv, Huffman, adaptive Huffman) can hide the complications and are sometimes implemented in hardware, as for tape- or disk-drive compression. Such schemes might be applied to analysis-format data for storage of simulations data, but estimates of savings are not available.

Some minor aspects of the proposed format were chosen to minimize the variety of bit patterns within a given table—improving their potential compressibility. However, no tests have been carried out.

Overhead in STAR tppad/tppixel Format

The tppad/tppixel format now used for TPC Slow Simulator (**tss**) output and TPC CLuster finder (**tcl**) input is both storage- and CPU-inefficient: redundant information and empty space are considerable, and bit-level decoding is required. An estimate from SN0282 of its storage overhead is reproduced here.

Each pad is indexed with four long int's, and each pixel takes an additional long int:

```
struct tss_tppad{
    long       jpix;       /* offset into the tppixel table */
    long       nseq;       /* number of sequences for this pad */
    long       secpad;     /* unique pad id within a sector */
    long       tpc_row;    /* encoded sector-row = 100*sector + row */
};

struct tss_tppixel{
    long       datum;      /* encoded tzero, nbins, adc values */
};
```

Total storage for a central Au-Au event ("standard" assumptions) is

$$(4 \cdot 4 + 4N_{\rm seq}L_{\rm seq})N_{\rm pads} = (16 + 4 \cdot 4 \cdot 8) \cdot 136,560 = 19,664,640 \text{ bytes} .$$

Since the ADC is allocated 10 bits in this format, the ADC data occupies 5,462,400 bytes, leaving 14,202,240 bytes of overhead—10.2× as much as the proposed format.

Application Considerations

Despite the preceding discussions, the most important questions about the TPC data structures do not concern CPU and storage overhead, but, rather, are they easily used for real applications? How do they affect the design of those applications? To answer this adequately requires actual coding and testing, but a few, simple *gedanken* experiments can be done here.

Event Server Considerations

It is foreseen that an event server will provide data consumers with TPC data subsets, so the designs of the event server and TPC data structures are inter-related, involving several issues:

1. Granularity supported by the event server. This is undecided, but should not be below the pad, or above the sector, level. At the pad level, a transaction would be dominated by the cost of extraction; at the sector level, there would usually be a significant I/O cost, as well. Information on a single pad is of limited value, but one can easily imagine wanting data below the sector level—a single readout board or FEE card, for example.

   Suggestion: Support data requests at the granularity of a FEE card or half pad row.

2. Form of events stored in the server. If events are stored in DAQ's format and translated separately for each data request, it is always cheaper to translate and send a smaller data unit. This scheme would be justified only if most requests are for small subsets of an event.

On the other hand, if DAQ data are translated and stored in the analysis format, the cost of extracting data must be balanced against the cost of transmitting unneeded data.

Suggestion: Store translated events to reduce latency, and define the event format to minimize the cost of extracting data at the supported granularity.

3. Event re-access through the event server. Events could be cached on the event server, allowing a user to access additional data subsets as needed, until s/he requests another event. This is practical if, and (probably) only if, the number of clients for data is less that the number of events that can be simultaneously stored on the server. The system would allow a user to be conservative about the volume of data requested interactively: less I/O, at the expense of more event storage.

   Suggestion: Desireable, if practical...

4. Simultaneous support for mapped and native data. DAQ probably does not usually transmit data from blind FEE channels, and would have to be told explicitly to do so, but this would be helpful for diagnosing and studying the electronics. If DAQ does transmit this information, it can be provided only in the native format to a consumer. This raises the question of whether the event server should simultaneously support native and mapped events or, alternatively, who translates into the mapped format?

   Suggestion: When producing native-format data, simultaneously produce and store the mapped-format data.

5. Level of request complexity supported by the event server. Supporting data granularity below the full TPC raises a question: what data combinations may be requested? If granularity is at the level of a half pad row, can one request all data from sectors 13–24, rows 44 and 45, for example?

   Suggestion: Support all possible combinations of legal subsets.

6. Form of data requests to the event server. The appropriate choice depends upon other design choices—supported granularity, complexity of requests, etc. To be clear: requests for data only at the sector level can be supported trivially, but describing an arbitrary data subset requires a complexity approaching that of a data format.

   Suggestion: An interesting idea would be to make the request in the form of the TPC data structure to be filled.

7. Redundant indexing tables. The **raw_pad_off_*** or **raw_chn_off_*** tables are easy to calculate as data are formatted and add only 0.52 MB (3%) more storage overhead to Au-Au events.

Suggestion: Always provide them.

On-line Extraction of Data Subsets

For the sake of discussion, assume that the on-line event server incorporates the above suggestions. Also assume that, at some given time, DAQ is providing mapped, zero-suppressed data from all sectors; the event server is producing mapped "grey" data (data on all pads), along with the optional **raw_pad_off_*** tables; and that an appropriate Staf directory structure exists (maybe passed to the server as the data request). Consider the process of extracting some data subsets:

Sector N, outer subsector: This is easy; most tables correspond to exactly one subsector. Copy all **TPC_DATA/Sector_N/*_out** tables, as well as any tables for RDO's 3–6; update all entries in **raw_sector_des**.

Sector N, pad rows 44, 45: This is worse. We copy the rows from **raw_row_idx_out** for pad rows 44 and 45, updating the indices in them. Next, from the original table, we take the starting indices and the number of rows corresponding to the two half pad rows for each of the other tables, and copy the appropriate parts of them. We probably still copy any tables for RDO's 3–6. Finally, update all entries in **raw_sector_des**, Row N that pertain to the sector, outer subsector, or RDO's 3–6. (The **raw_pad_off_out** table contains relative offsets, not absolute row numbers, so no update is needed there.)

Sector N, pad row 44, pads 70, 71, 72: This is bad, but tractable, if the **raw_pad_off_out** table exists. We jump to the relevant data, and can use **raw_row_idx_out** and **raw_pad_off_out** to figure out the row ranges in the sequence and pixel tables, if we're careful; then, copy them. We have to update many of the entries in **raw_sector_des**, **raw_row_idx_out** and **raw_pad_off_out**, then create and fill **raw_pad_id_out**.

This probably is *possible*, but more microscopic than should be expected from an event server, philosophically; the suggestion for half-pad-row granularity stands.

Sector N, RDO 6: This request would probably return an error code to the user, since we assumed that the event server is producing mapped data, so that data from blind channels would be unavailable.

Data Pre-processing Considerations

In general, before cluster and hit finding, the data must be pre-processed: 8-to-10-bit uncompression; pedestal subtraction; gain correction; threshold application; zero suppression; zero re-suppression after applying a threshold; and/or embedding Monte Carlo tracks into real data. We examine these problems briefly to determine whether or not our structures are compatible with practical solutions.

One-to-One Table Mapping Problems

In the simplest case, each entry in a table is mapped into entries in a parallel table by a defined function; an obvious example is 8-to-10-bit uncompression. Since Staf doesn't support 10-bit integers, the translation would be from u_char's to u_short's. One would get the row count for the 8-bit table from Staf and allocate a table for the u_short's, then loop over the rows, translating into the new table. At the end, the old, 8-bit table could be deleted. No operations on higher level tables are needed.

Data Remapping within a Table

Entries within a table may have to be assigned new values. In the simplest case, a fixed function is applied to a table, without forcing any reformatting of its contents. An example would be the application of a fixed threshold to "black" data. One gets the table's row count from Staf, then loops over the rows, setting sub-threshold values to zero. No higher level tables have to be accessed.

More work is required if the mapping function has dependencies, as would a gain correction or pedestal subtraction. Then, one must work through the table using the higher level structures to associate the table entry to a particular pad or pixel. Simultaneously, one must work through another table structure to get the applicable gain or pedestal. In principle, dependent information could be put into tables parallel to the ones being modified, but this would usually just shift the same work to a different place.)

Even more work is required when the mapping function obsoletes the structure; for example, applying a threshold usually results in time-bucket sequences containing zeroes. For a cluster finder like **tcl**, this is not a problem, since sequences are copied into a flat space before operations. However, a more efficient cluster finder would operate directly on the found sequences without examining contents. Then, sequences would have to be shortened or split before cluster finding, forcing significant shuffling of the data. If one is, in effect, zero-suppressing "black" data, even **raw_sector_des** might have to be edited.

Cluster-finding Considerations

Currently, the **tcl** cluster finder uses the tppad/tppixel structures only for storage; data are copied to working arrays before any operations. Almost any format can replace tppad/tppixel for this. However, to make **tcl** efficient, it should not move data unnecessarily, or ignore existing information.

Zero-suppressed sequences should be identified and stored in the mapped data format, and this will be the usual form of data from the on- or off-line event server, after translation. One would also create the **raw_pad_off_*** table of pad-level index offsets, if it didn't already exist, to allow easy, pad-level access to the data. One would also create a table, parallel to **raw_seq_des_***, in which to store cluster ID's assigned to the sequences. The actual cluster finding would be done with something like the "peep-hole cluster finder.[5]"

[5] Dirk Schmischke, *Documentation for the new peep-hole cluster finder,*

The pattern-recognition problem is solved at this point, but the clusters are not easily accessible as such. They could be stored in arrays, as **tcl** does, but a framework analogous to the data-distribution structures proposed in this note would be more efficient. This requires a new tree in **Sector_N/**, but this could point back to the original tree at the sequence level, as in Fig. 5, avoiding data copying. Alternatively, if heavy cluster processing is expected, the sequences could be sorted into the cluster tree, as in Fig. 6, optionally deleting the original raw data tree.

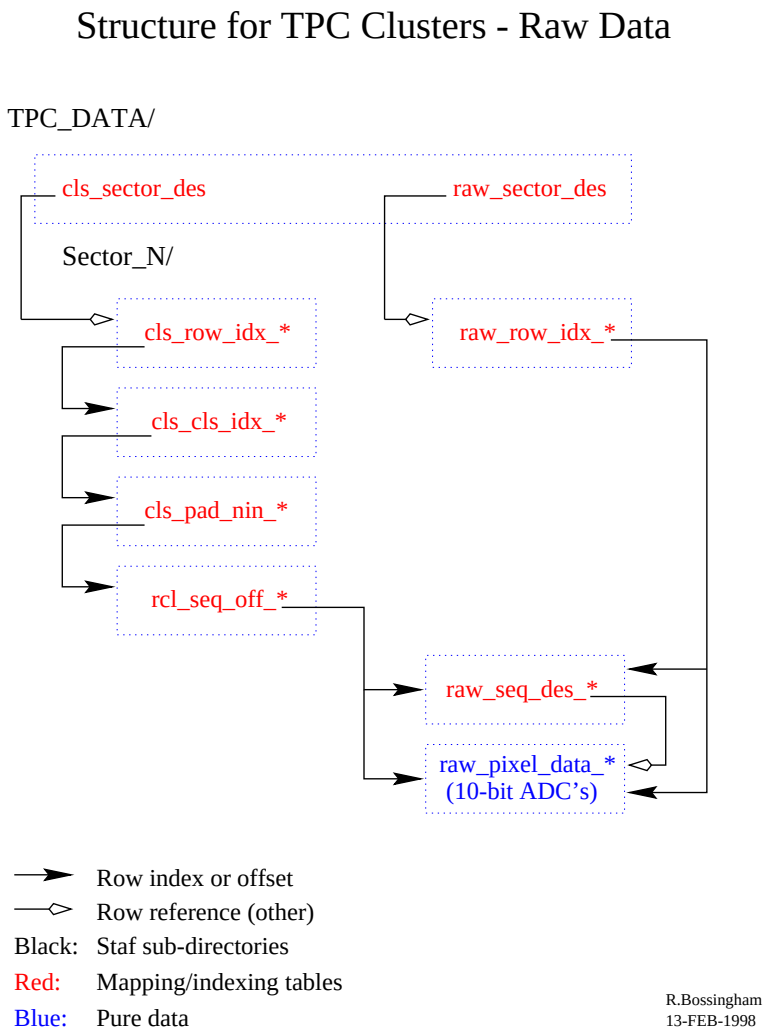

Figure 5: Schematic TPC cluster represention pointing to "raw" time-bucket sequences. The cluster tables will be explicitly defined in a separate note.

http://www.rhic.bnl.gov/star/starlib/doc/www/html/daq_l/public/software/cluster.html .

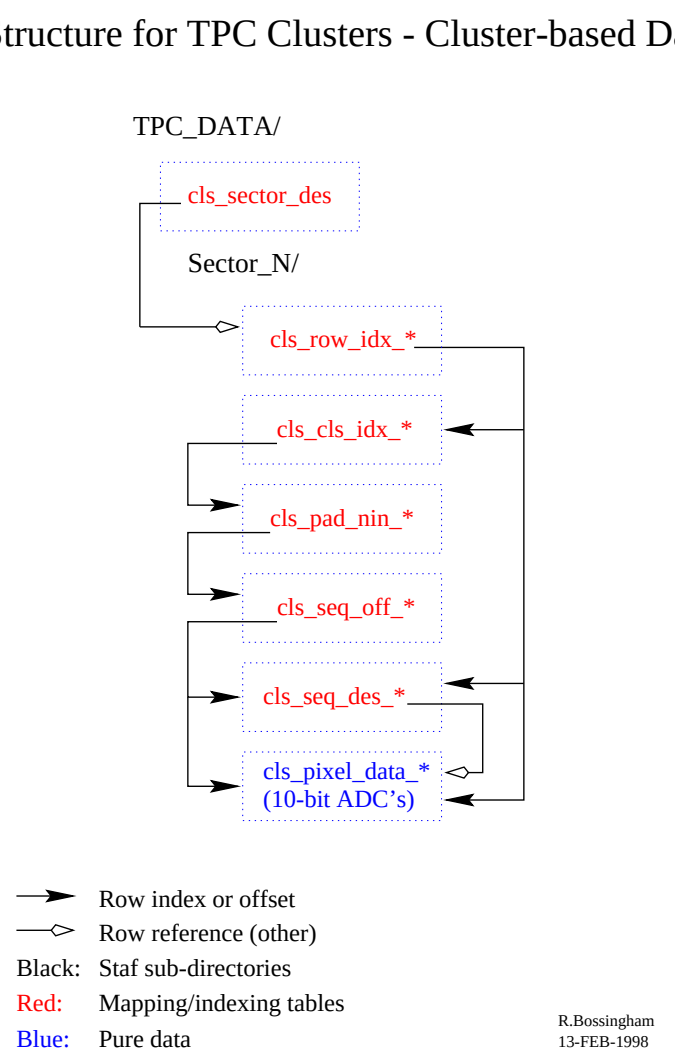

Figure 6: Schematic TPC cluster represention pointing to time-bucket sequences sorted by cluster. The cluster tables will be explicitly defined in a separate note.

Hit-finding Considerations

In general, there is not a 1:1 correspondence between clusters and hits; some clusters contain multiple hits, some contain none. Also, conventional ("follow-your-nose") track finding in an environment of dense hits requires some artificial structuring so that each search for a hit need not go through the entire hit list for a pad row. This complicates the structure and motivates a separate data tree.

The cluster and hit data structures should also allow a second pass (not yet implemented) in which global information from tracking iteratively improves hit finding. If tracks are directly associated only with hits, this implies the need for pointers from hits to clusters, and from clusters to hits.

The obvious structure for hits within a pad row is a cartesian grid (pad row vs. drift direction), grouping the hits within each grid element. This is the scheme assumed in Fig. 7. Ideally, the grid elements would be square, minimizing the perimeter-to-area ratio, as well as large, compared to the maximum distance at which a hit could be assigned to a track, and compared to the typical separation between hits in a Au-Au event to reduce overhead. On the other hand, a grid element should be small enough that it holds only a small fraction of the hits on the pad row, so that the gained efficiency justifies the complications.

The main complication occurs near the perimeter of a grid element, where hits in multiple grid elements can lay within the search radius around a track. A straight-forward approach would verify that the grid was large compared to the search radius, and then search the multiple (up to four) groups of hits.

A more elaborate scheme, due to Iwona Sakrejda, uses overlapping elements, with the overlap defined by the maximum search radius, allowing hits to be assigned to multiple groups. This approach is simpler, provided that one groups, not hits, but *pointers* to the hits, so that there is only a single copy of the hit table and track assignments.

Conclusions

The table structures proposed in this note are compatible with STAR's needs for event serving, cluster/hit finding, and two-pass tracking, and should facilitate their implementation or improvement. However, to be efficient, the software must exploit the structures—data structures are meaningless in a vacuum. Software requirements and design could easily be set so as to force modification of the structures, or even to preclude their effective use.

Before final adoption of any structures, it strongly recommended that the event server be specified, and that the plan for reconstruction software be reviewed. In addition, some representative, preliminary code using the structures should be written and benchmarked to verify usability and efficiency, as well as to check for holes in the definition (some implementation details have clearly not been defined).

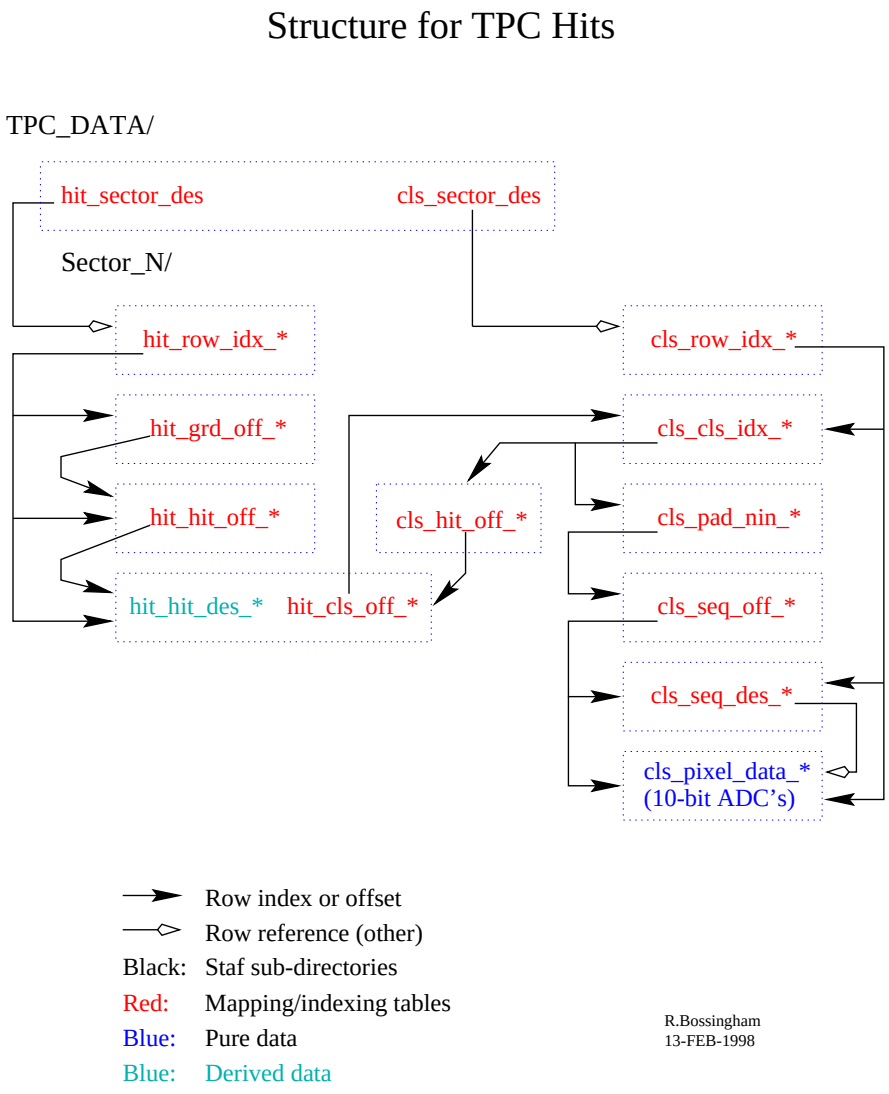

Figure 7: Schematic TPC hit represention, with tables relating hits to clusters, and vice versa. The hit and cluster tables will be defined explicitly in a separate note.